UTP

project III

December 6, 2024

General assumptions

In the classes contained in the package models simulation models are defined, performing some calculation for periods (eg. years) A definition of a model contains:

1. A field named LL, which is the number of years of simulation;
2. Fields which are variables of the model - those are arrays of double;
3. Optionally some auxiliary fields;
4. A method: public void run(), which performs calculation;
5. Optionally other auxiliary methods.

Fields 1 and 2 are marked with the `@Bind` annotation, which allows:

- assigning values to variables of input model before performing calculations,
- retrieve the variable values listed in the model (after the calculation).

All variables annotated with `@Bind` are available for scripts that can be run after the model calculations and do some further calculations. They are also available for other models.

To manage calculations, a class Controller is introduced, which has the following public components:

- A constructor - Controller(String modelName) - model class name is a parameter,
- Controller readDataFrom(String fname) - reads the data for the calculation of a file named fname.
- Controller runModel() - launches model calculations,
- Controller runScriptFromFile(String fname) - executes a script from a file named fname,
- Controller runScript(String script) - executes the script code specified as a string,
- String getResultsAsTsv() - returns the calculation results (all variables in the model and variables created in the script) as a string, the subsequent rows contain the variable name and its value, separated by tabs.

Assumptions about the input data

Files with the input data include the following lines in the form of:

```
variable_name value1 [ value2 ... valueN ] }
```

A special row that begins with the word LATA specifies the period of calculation, for example.

LATA 2015 2016 2017 2018 2019

Based on this row, the value of the special variable LL is defined by (the number of years the calculation) which is available in the model and in scripts.

A number of the values for the variables can be from 1 to LL. If they are less than LL, the rest are determined on the last of these values.

Example

We have the following model:

```
public
    class Model1 {

    @Bind private int LL; // number of years

    @Bind private  double[] twKI; // the growth rate of private consumption
    @Bind private  double[] twKS; // the growth rate of public consumption
    @Bind private  double[] twINW; // investment growth
    @Bind private  double[] twEKS; // export growth
    @Bind private  double[] twIMP; // import growth

    @Bind private double[] KI;  // private consumption
    @Bind private double[] KS;  // public consumption
    @Bind private double[] INW;  // investments
    @Bind private double[] EKS;  // export
    @Bind private double[] IMP;  // import
    @Bind private double[] PKB;  // GDP

    private double temp; // this field is not associated with the data model or with the results

    public Model1() {}

    public void run() {
        PKB = new double[LL];
        PKB[0] = KI[0] + KS[0] + INW[0] + EKS[0] - IMP[0];
        for (int t=1; t < LL; t++) {
          KI[t] = twKI[t]* KI[t-1];
          KS[t] = twKS[t]* KS[t-1];
          INW[t] = twINW[t]* INW[t-1];
          EKS[t] = twEKS[t]* EKS[t-1];
          IMP[t] = twIMP[t]* IMP[t-1];
          PKB[t] = KI[t] + KS[t] + INW[t] + EKS[t] - IMP[t];
        }
    }
}
```

For a file data1.txt with the following form:

```
LATA   2015 2016 2017 2018 2019
twKI   1.03
twKS   1.04
twINW  1.12
twEKS  1.13
twIMP  1.14
```

```
KI      1023752.2
KS      315397
INW     348358
EKS     811108.6
IMP     784342.4
```

the following excerpt of the main program:

```
Controller ctl = new Controller("Model1");
ctl.readDataFrom(dataDir + "data1.txt")
                .runModel()
String res= ctl.getResultsAsTsv();
System.out.println(res);
```

will print on the console:

```
LATA    2015    2016    2017    2018    2019
twKI    1.03    1.03    1.03    1.03    1.03
twKS    1.04    1.04    1.04    1.04    1.04
twINW   1.12    1.12    1.12    1.12    1.12
twEKS   1.13    1.13    1.13    1.13    1.13
twIMP   1.14    1.14    1.14    1.14    1.14
KI      1023752.2   1054464.766 1086098.70898   1118681.6702494002  1152242.1203568822
KS      315397.0    328012.88   341133.3952     354778.73100800003  368969.88024832006
INW     348358.0    390160.96   436980.2752000001   489417.90822400013  548148.0572108802
EKS     811108.6    916552.7179999999   1035704.5713399998  1170346.1656141996  1322491.1671440455
IMP     784342.4    894150.3359999999   1019331.3830399998  1162037.7766655996  1324723.0653987834
PKB     1714273.4   1795040.9880000001  1880585.5676800003  1971186.6984300003  2067128.1595613444
```

and for a file data2.txt, which looks like this:

```
LATA    2015 2016 2017 2018 2019
twKI    1.03 1.05 1.07
twKS    1.04
twINW   1.12
twEKS   1.13
twIMP   1.14
KI      1023752.2
KS      315397
INW     348358
EKS     811108.6
IMP     784342.4
PKB     1714273.4 1815516.032 1944672.4554000003 2083203.6166496002 2231733.528866293
```

we get after

```
ctl.readDataFrom(dataDir + "data2.txt").runModel()
```

such a result:

```
LATA    2015    2016    2017    2018    2019
twKI    1.03    1.05    1.07    1.07    1.07
twKS    1.04    1.04    1.04    1.04    1.04
twINW   1.12    1.12    1.12    1.12    1.12
twEKS   1.13    1.13    1.13    1.13    1.13
twIMP   1.14    1.14    1.14    1.14    1.14
KI      1023752.2   1074939.81  1150185.5967    1230698.5884690003  1316847.4896618305
KS      315397.0    328012.88   341133.3952     354778.73100800003  368969.88024832006
INW     348358.0    390160.96   436980.2752000001   489417.90822400013  548148.0572108802
EKS     811108.6    916552.7179999999   1035704.5713399998  1170346.1656141996  1322491.1671440455
IMP     784342.4    894150.3359999999   1019331.3830399998  1162037.7766655996  1324723.0653987834
PKB     1714273.4   1815516.032 1944672.4554000003  2083203.6166496002  2231733.528866293
```

Scripts

Scripts can be written in any script mechanism eg. Groovy, Python ect. In the scripts, all loaded data and the data from the model should be available (but only these marked

by @Bind). Variables created in the script should be available both in the resulting tsv, as well as in other models (if there is any field marked by @Bind), as well as other scripts. The exception to this rule are variables with single-letter names written in lowercase (eg. i, j, k, p) - these variables do not appear in the results and won't be available in models or other scripts.

A example script stored in the file script1.groovy might look like this (here is calculated indicator of export capacity):

```
ZDEKS = new double[LL]
for (i = 0; i < LL; i++) {
 ZDEKS[i] =  EKS[i]/PKB[i];
}
```

Attention: in this script a variable LL is available which denotes the number of years, and also there are variables resulting from model calculations (PKB, EKS). The script creates new variables ZDEKS and i, but only ZDEKS will be visible in the results and available for other models and scripts.

An example invocation:

```
Controller ctl = new Controller("Model1");
ctl.readDataFrom(dataDir + "data2.txt")
                .runModel()
                .runScriptFromFile(scriptDir + "script1.groovy");
String res= ctl.getResultsAsTsv();
System.out.println(res);
```

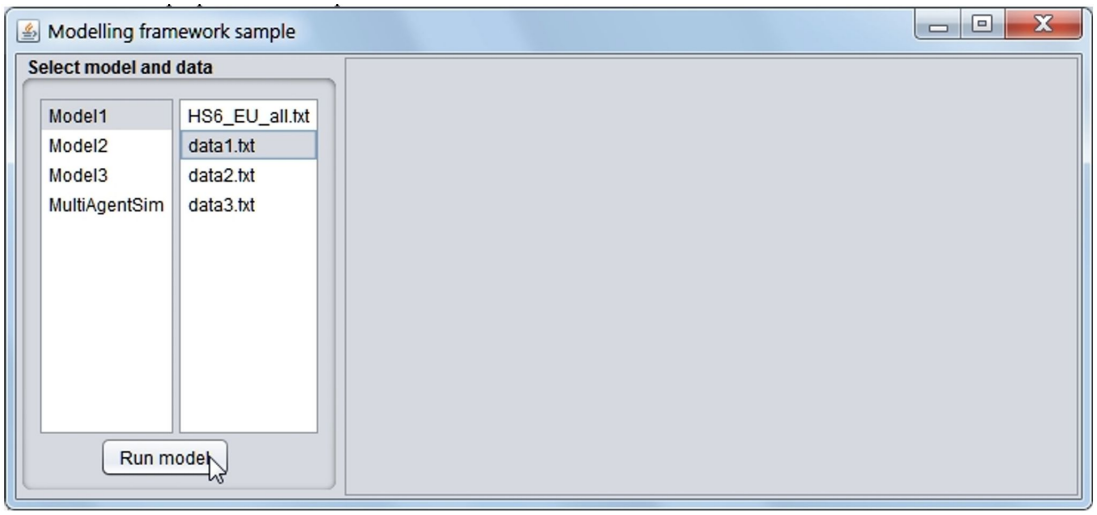
will print out:

```
LATA    2015    2016    2017    2018    2019
twKI    1.03    1.05    1.07    1.07    1.07
twKS    1.04    1.04    1.04    1.04    1.04
twINW   1.12    1.12    1.12    1.12    1.12
twEKS   1.13    1.13    1.13    1.13    1.13
twIMP   1.14    1.14    1.14    1.14    1.14
KI      1023752.2   1074939.81  1150185.5967    1230698.5884690003  1316847.4896618305
KS      315397.0    328012.88   341133.3952     354778.73100800003  368969.88024832006
INW     348358.0    390160.96   436980.2752000001   489417.90822400013  548148.0572108802
EKS     811108.6    916552.7179999999   1035704.5713399998  1170346.1656141996  1322491.1671440455
IMP     784342.4    894150.3359999999   1019331.3830399998  1162037.7766655996  1324723.0653987834
PKB     1714273.4   1815516.032 1944672.4554000003  2083203.6166496002  2231733.528866293
ZDEKS   0.47315008212808995 0.5048441885640148 0.5325856127925489 0.561801139485567 ↩
    0.5925847105123984
```

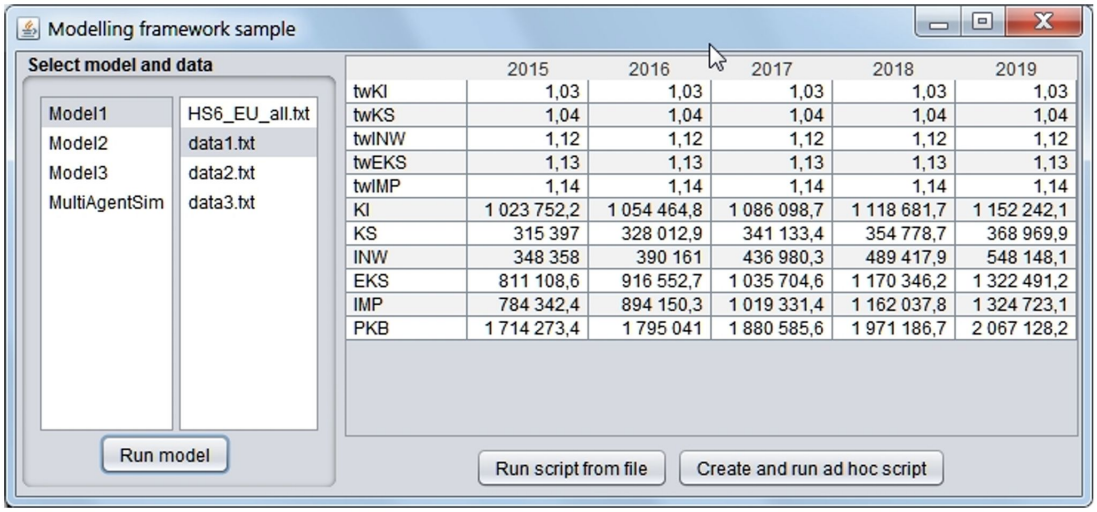
GUI

Integral part of a project is a GUI to work with models and scripts.

Here's an ad hoc prepared example of such an interface.



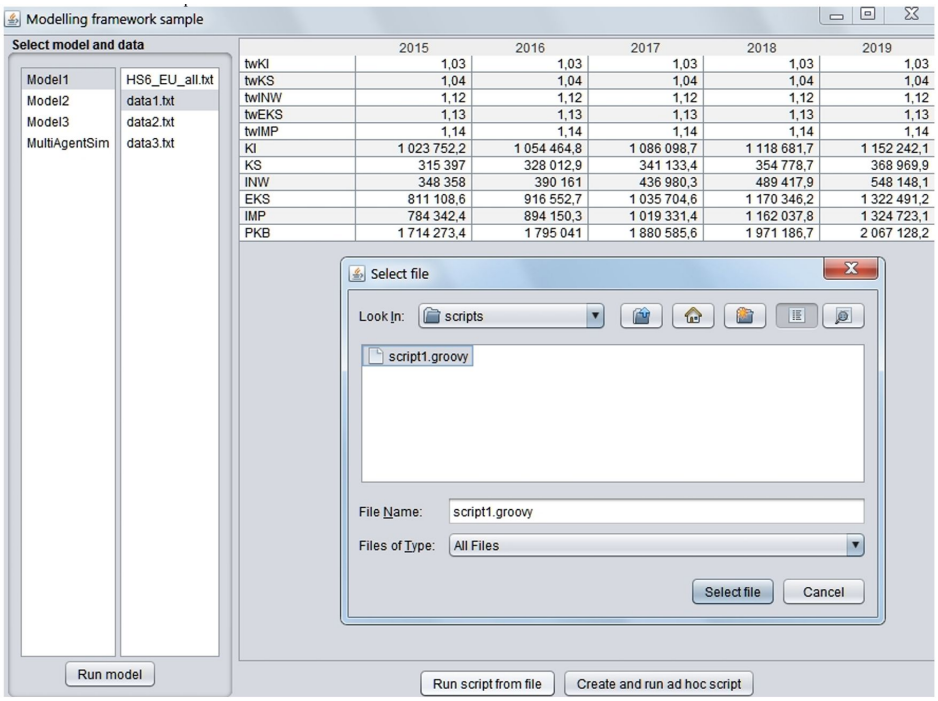
When you press the "Run model" button:



	2015	2016	2017	2018	2019
twKI	1,03	1,03	1,03	1,03	1,03
twKS	1,04	1,04	1,04	1,04	1,04
twINW	1,12	1,12	1,12	1,12	1,12
twEKS	1,13	1,13	1,13	1,13	1,13
twIMP	1,14	1,14	1,14	1,14	1,14
KI	1 023 752,2	1 054 464,8	1 086 098,7	1 118 681,7	1 152 242,1
KS	315 397	328 012,9	341 133,4	354 778,7	368 969,9
INW	348 358	390 161	436 980,3	489 417,9	548 148,1
EKS	811 108,6	916 552,7	1 035 704,6	1 170 346,2	1 322 491,2
IMP	784 342,4	894 150,3	1 019 331,4	1 162 037,8	1 324 723,1
PKB	1 714 273,4	1 795 041	1 880 585,6	1 971 186,7	2 067 128,2

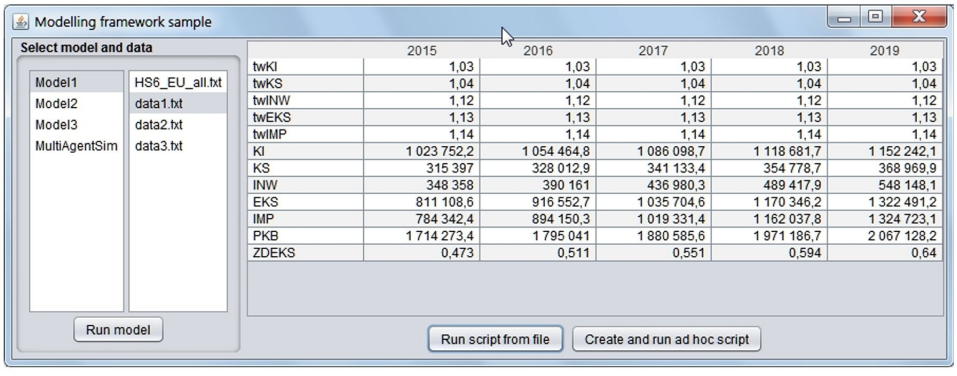
Note: Please pay attention to the formatting of numbers

You can now choose a script from the file:



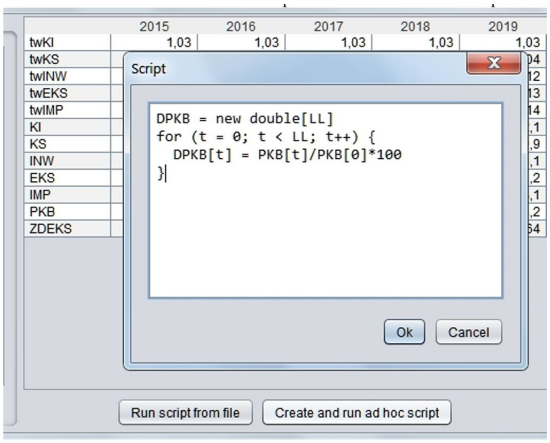
	2015	2016	2017	2018	2019
twKI	1,03	1,03	1,03	1,03	1,03
twKS	1,04	1,04	1,04	1,04	1,04
twINW	1,12	1,12	1,12	1,12	1,12
twEKS	1,13	1,13	1,13	1,13	1,13
twIMP	1,14	1,14	1,14	1,14	1,14
KI	1 023 752,2	1 054 464,8	1 086 098,7	1 118 681,7	1 152 242,1
KS	315 397	328 012,9	341 133,4	354 778,7	368 969,9
INW	348 358	390 161	436 980,3	489 417,9	548 148,1
EKS	811 108,6	916 552,7	1 035 704,6	1 170 346,2	1 322 491,2
IMP	784 342,4	894 150,3	1 019 331,4	1 162 037,8	1 324 723,1
PKB	1 714 273,4	1 795 041	1 880 585,6	1 971 186,7	2 067 128,2

and immediately see additional results:

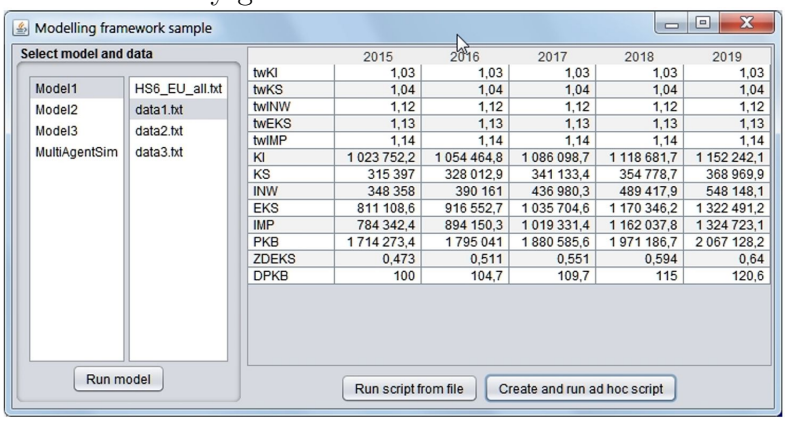


	2015	2016	2017	2018	2019
twKI	1,03	1,03	1,03	1,03	1,03
twKS	1,04	1,04	1,04	1,04	1,04
twINW	1,12	1,12	1,12	1,12	1,12
twEKS	1,13	1,13	1,13	1,13	1,13
twIMP	1,14	1,14	1,14	1,14	1,14
KI	1 023 752,2	1 054 464,8	1 086 098,7	1 118 681,7	1 152 242,1
KS	315 397	328 012,9	341 133,4	354 778,7	368 969,9
INW	348 358	390 161	436 980,3	489 417,9	548 148,1
EKS	811 108,6	916 552,7	1 035 704,6	1 170 346,2	1 322 491,2
IMP	784 342,4	894 150,3	1 019 331,4	1 162 037,8	1 324 723,1
PKB	1 714 273,4	1 795 041	1 880 585,6	1 971 186,7	2 067 128,2
ZDEKS	0,473	0,511	0,551	0,594	0,64

or choose the "Create and run ad-hoc script" to enter the code of the script:



and immediately get the result:



	2015	2016	2017	2018	2019
twKI	1,03	1,03	1,03	1,03	1,03
twKS	1,04	1,04	1,04	1,04	1,04
twINW	1,12	1,12	1,12	1,12	1,12
twEKS	1,13	1,13	1,13	1,13	1,13
twIMP	1,14	1,14	1,14	1,14	1,14
KI	1 023 752,2	1 054 464,8	1 086 098,7	1 118 681,7	1 152 242,1
KS	315 397	328 012,9	341 133,4	354 778,7	368 969,9
INW	348 358	390 161	436 980,3	489 417,9	548 148,1
EKS	811 108,6	916 552,7	1 035 704,6	1 170 346,2	1 322 491,2
IMP	784 342,4	894 150,3	1 019 331,4	1 162 037,8	1 324 723,1
PKB	1 714 273,4	1 795 041	1 880 585,6	1 971 186,7	2 067 128,2
ZDEKS	0,473	0,511	0,551	0,594	0,64
DPKB	100	104,7	109,7	115	120,6